(Kleine) Einführung zu Künstlicher Intelligenz in Brettspielen

Patrick Schäfer

patrick.schaefer@hu-berlin.de

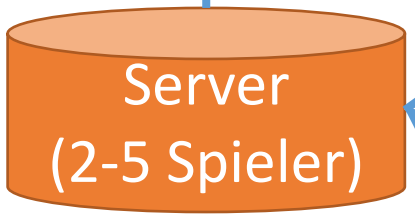
Agenda

- Today
  - Einführung zu Künstlicher Intelligenz in Brettspielen
- This week:
  - finalize tasks in your sprint backlog (incl. tests, code review)
- Next Monday, 13:30
  - Sprint #3 Review Meeting; bring a laptop & presentable prototype
- Monday, 17.12., 13:30
  - First Mile Stone Review Meeting; bring a laptop & presentable prototype

Anforderungskatalog

Playback

JSON



1. Meilenstein
Observer-GUI

2. Meilenstein
KI-Spieler /
Wettbewerb

Herangehensweise

- “Getting beaten by own code is the sweetest defeat”
  - Blog-Post zu Brettspiel-KI in Unity
    https://madewith.unity.com/stories/boardgame-ai

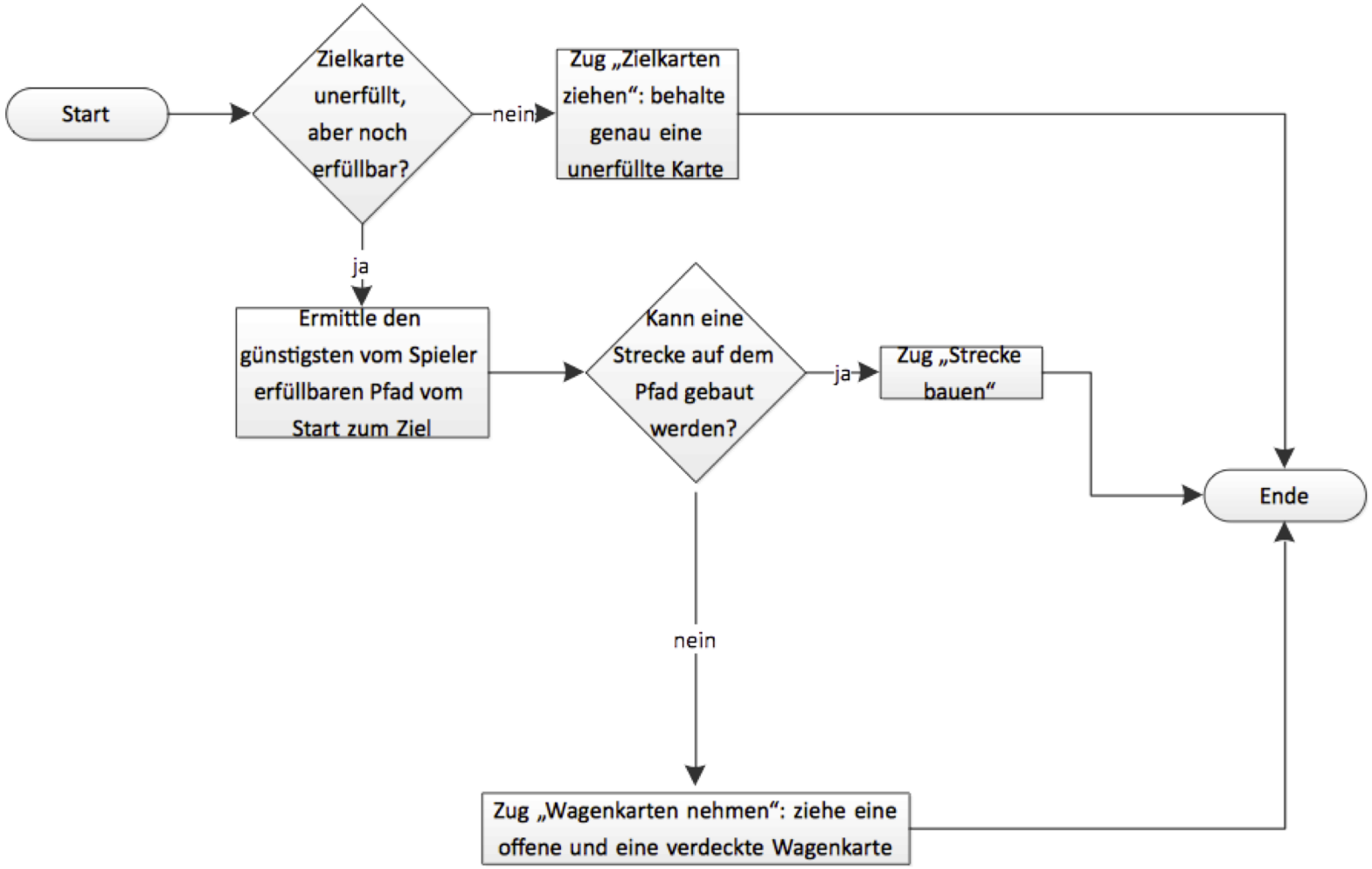
- Grundsätzliche Ansätze:
  - Regelbasiert
  - Minimax Tree Search
  - Monte-Carlo Tree Search
  - (Reinforcement Learning (Deep Learning))

Regelbasierter Ansatz

- Regelbasierter Ansatz:
  - Spielsituation analysieren, Regeln durchgehen
  - Wende Regel mit höchster Priorität an, deren Vorbedingung erfüllt ist
  - Wenn keine Regel greift, auf Standard-KI ausweichen

- Beispiel:
  - Prio 10: Zug „Zielkarten ziehen“
    - Vorbedingung: Alle Zielkarten auf der Hand sind erfüllt.
    - Aktion: Behalte neue Zielkarte(n) mit minimalen (Bau-)Kosten

Einfache KI


Start
Zielkarte unerfüllt, aber noch erfüllbar?
nein
Zug „Zielkarten ziehen“: behalte genau eine unerfüllte Karte
ja
Ermittle den günstigsten vom Spieler erfüllbaren Pfad vom Start zum Ziel
Kann eine Strecke auf dem Pfad gebaut werden?
ja
Zug „Strecke bauen“
Ende
nein
Zug „Wagenkarten nehmen“: ziehe eine offene und eine verdeckte Wagenkarte

Regelbasierter Ansatz

- Prio 100: Zug „Strecke bauen“
  - Aktion: Baue Strecke auf einem günstigsten Pfad einer Zielkarte
  - Vorbedingung: der Spieler hat passende Wagenkarten und mindestens eine nicht erfüllte Zielkarte auf der Hand
- Prio 50: Zug „Wagenkarte nehmen“
  - Aktion: Ziehe (zwei) passende Wagenkarte(n)
  - Vorbedingung: Es kann keine Strecke gebaut werden und es gibt nicht erfüllte Zielkarten
- Prio 10: Zug „Zielkarten ziehen“
  - Aktion: Behalte neue Zielkarte(n) mit minimalen (Bau-)Kosten
  - Vorbedingung: Alle Zielkarten auf der Hand sind erfüllt.
- Default: Ziehe zwei verdeckte Wagenkarten

Gestaltungsmöglichkeiten

- Zug: Strecke bauen
  - Was sind die günstigsten Pfade (Steiner Tree, Dijkstra...)?
  - Haben wir den längsten Pfad (Longest Path)?
- Zug: Zielkarten ziehen
  - Sollten wir Zielkarten ziehen oder beendet ein anderer Spieler das Spiel?
  - Welche Zielkarten behalten wir (Steiner Tree, Dijkstra...)?
  - Welche Zielkarten sind erfüllt / nicht erfüllt (BFS, DFS)?
- Zug: Wagenkarten ziehen
  - Welche Wagenkarten brauchen wir?
- Mitspieler
  - Was machen die Mitspieler?
  - Sollte ich das Spiel beenden?

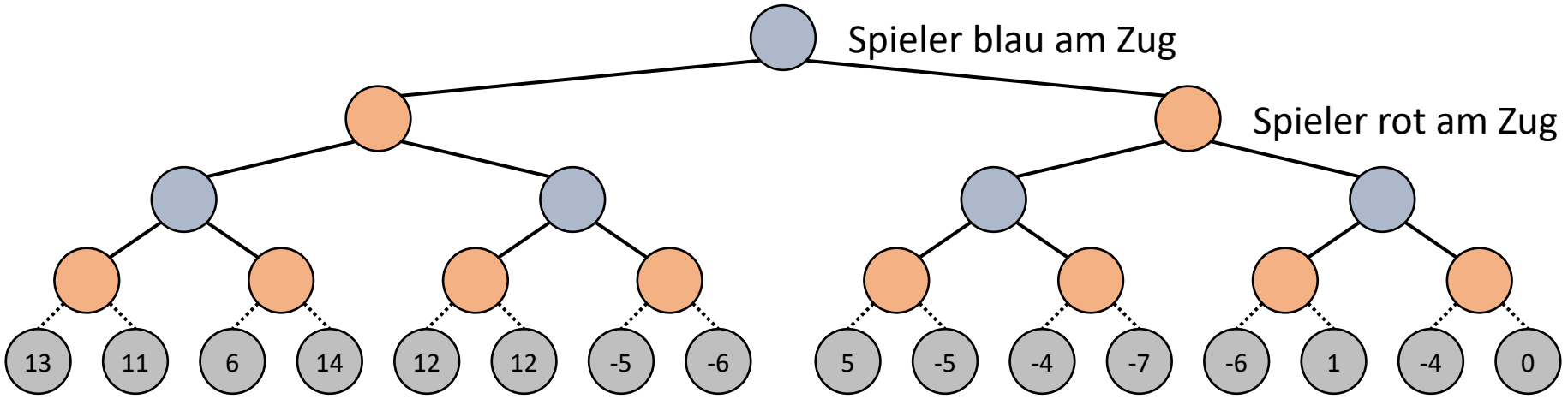
Entscheidungsbäume

- Es ist schwierig, den Zug der Mitspieler bei regelbasierten Ansätze zu modellieren

- Alternative Strategien:
  - Minimax
  - Monte-Carlo Tree Search

Entscheidungsbaum

- Spiele können als Baum modelliert werden:
  - Jeder Knoten repräsentiert einen Zustand des Spiels
  - Jede Kante repräsentiert einen möglichen Spielzug
  - Jedes Blatt repräsentiert ein mögliches Spielende und lässt sich bewerten
- Utility des Knotens entspricht (hier) Differenz des Punktestands



Entscheidungsbaum

- Verzweigungsgrad entspricht möglichen Spielzügen

- Beispiel:
  - Strecke bauen:
    alle möglichen Strecken, die mit Handkarten gebaut werden können
  - Wagenkarte ziehen:
    alle möglichen offene Farben ziehen und verdeckt ziehen
  - Zielkarten ziehen:
    für jede Zielkarte prüfen, ob sie erfüllt ist

Minimax

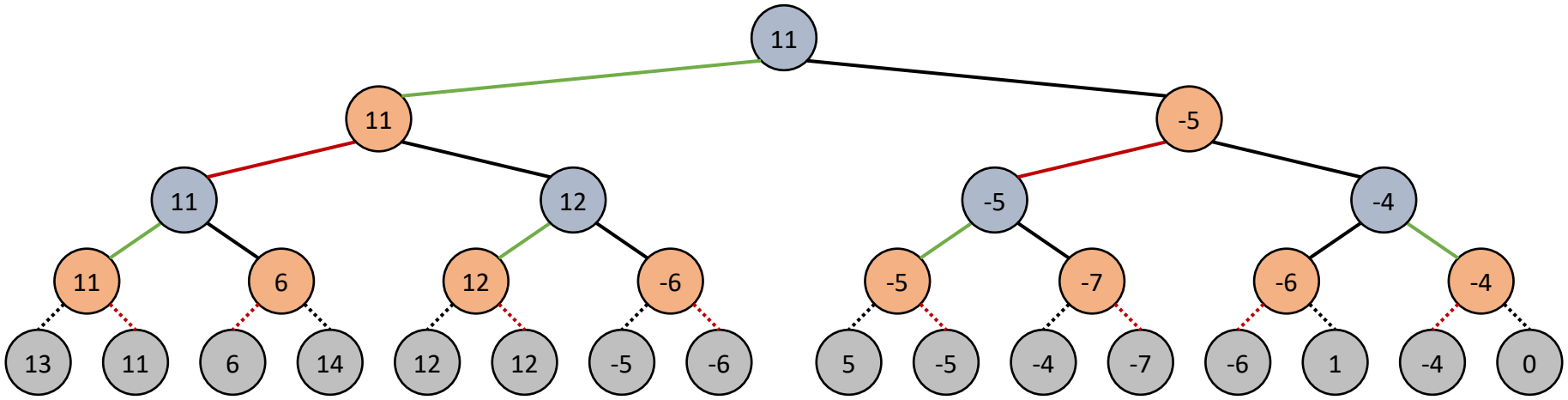
- Idee:
  - Wir spielen abwechselnd
  - Der Gegenspieler versucht meinen Gewinn zu minimieren

Spieler blau maximiert

Spieler rot minimiert

Spieler blau maximiert

Spieler rot minimiert

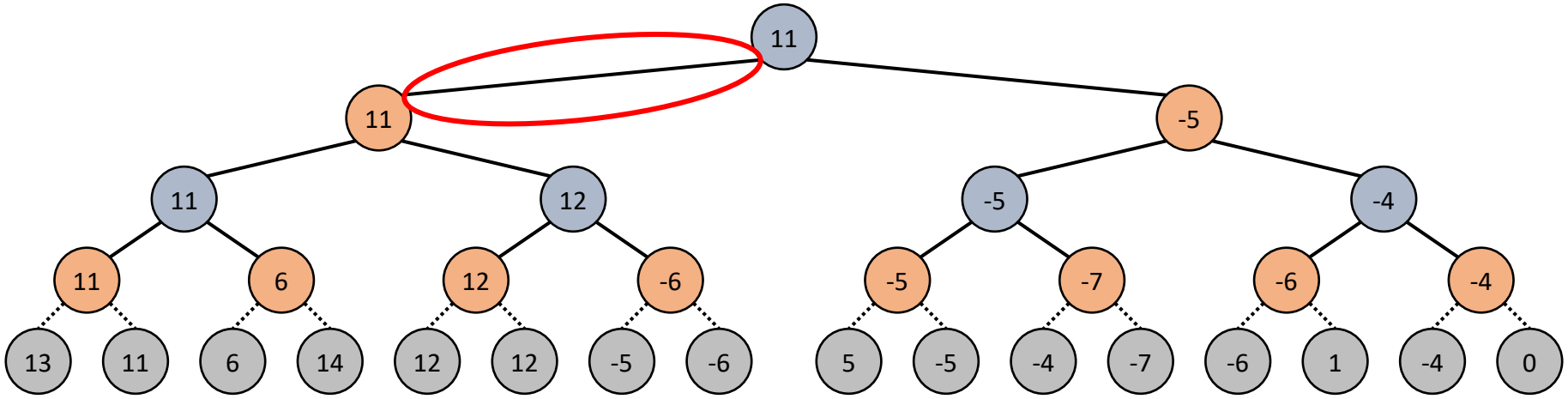


Minimax

- Idee: minimiere Chance zu Verlieren bei maximal guten Zügen des Gegners
- Vorgehensweise:
  - propagiere Bewertungen der Blätter von unten nach oben
  - alterniere zwischen Minimum und Maximum
  - wähle Kind der Wurzel mit höchstem Wert
- Problem: Bei k möglichen Zügen, ergibt sich nach n Spielzügen ein Baum der Größe $\mathrm{O}(k^n)$

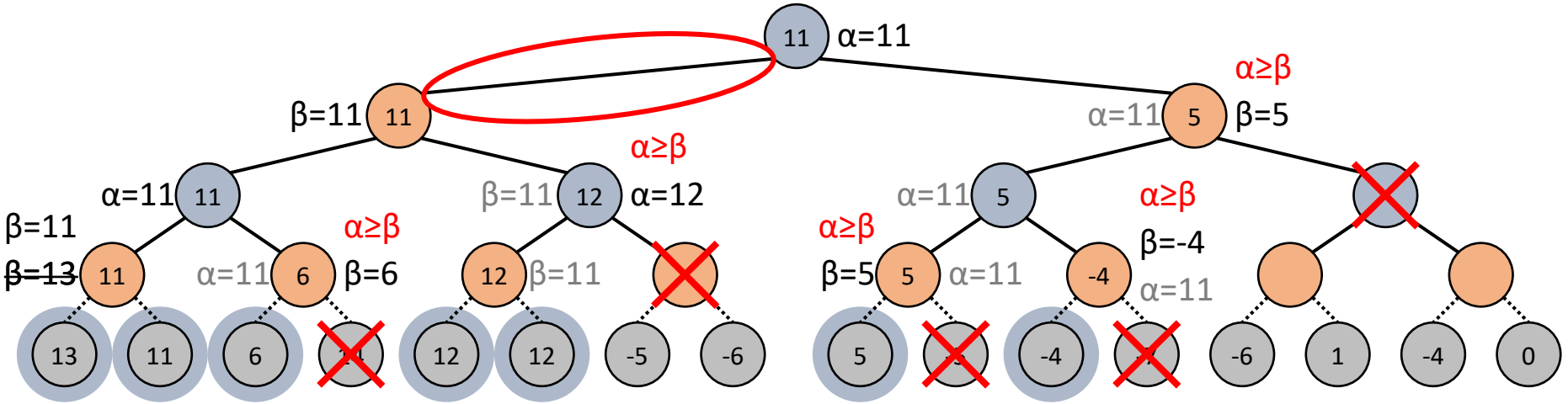
Spieler blau maximiert

Spieler rot minimiert



Optimierung: Alpha-Beta-Pruning

- Idee: Suboptimale Pfade brauchen wir nicht weiter zu betrachten
- Vorgehensweise:
  - Tiefensuche zu den Blättern, beginnend bei der Wurzel
  - Beim Backtracking werden Knoten alternierend mit α- und β-Werten versehen
  - α-Wert: maximaler von Spieler Blau garantierbarer Utility-Wert
  - β-Wert minimaler von Spieler Rot garantierbarer Utility-Wert
  - Beim Absteigen werden α- und β-Werte nach unten weitergereicht
  - Verwerfe Äste ab Knoten, für die gilt α≥β



Alpha-Beta Pruning

- Vorteil Pruning hat keinen Einfluss auf die Korrektheit

- Best-Case
  - Der beste Zug entspricht jeweils dem linkesten Knoten
  - Alle anderen Züge können verworfen werden
  - Idealerweise: Züge geeignet vorsortieren

- Worst-Case
  - Züge sind entsprechend der Güte aufsteigend sortiert
  - Alle Pfade müssen iteriert werden

Monte-Carlo Tree Search

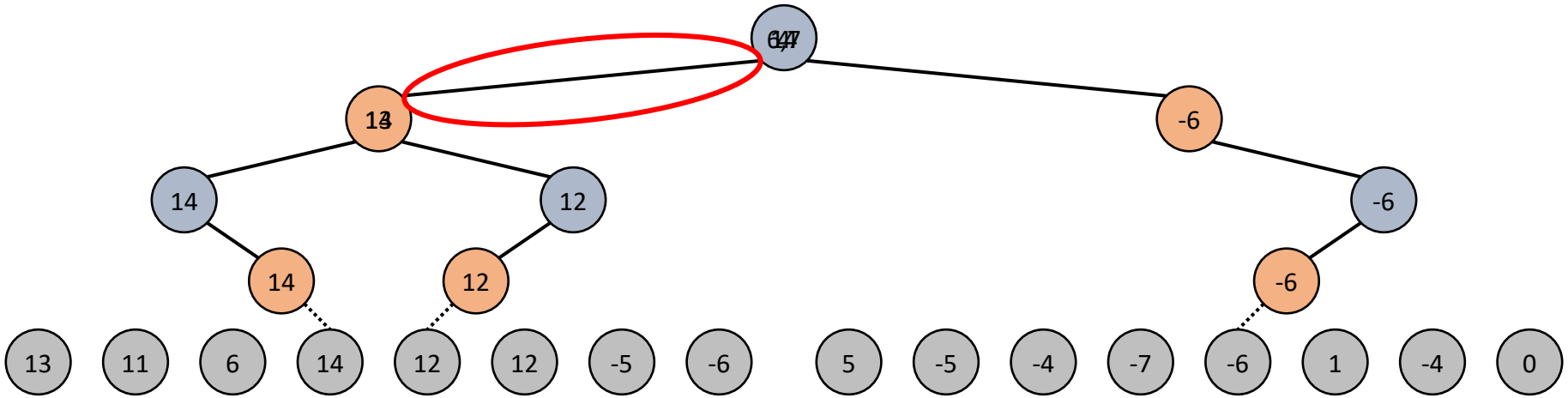
- Problem:
  - Der Entscheidungsbaum ist zu groß $O(10^{54})$, um alle Knoten zu iterieren

- Ansatz:
  - Iteriere Entscheidungsbaum durch zufällige Züge
  - Punktestand jedes Blattes wird genutzt, um Knoten auf dem Pfad zu gewichten
  - Knoten mit hoher Gewinnrate werden mit höherer Wahrscheinlichkeit erneut besucht

  - Zuletzt: Wähle den Knoten (Spielzug), der am höchsten gewichtet wurde

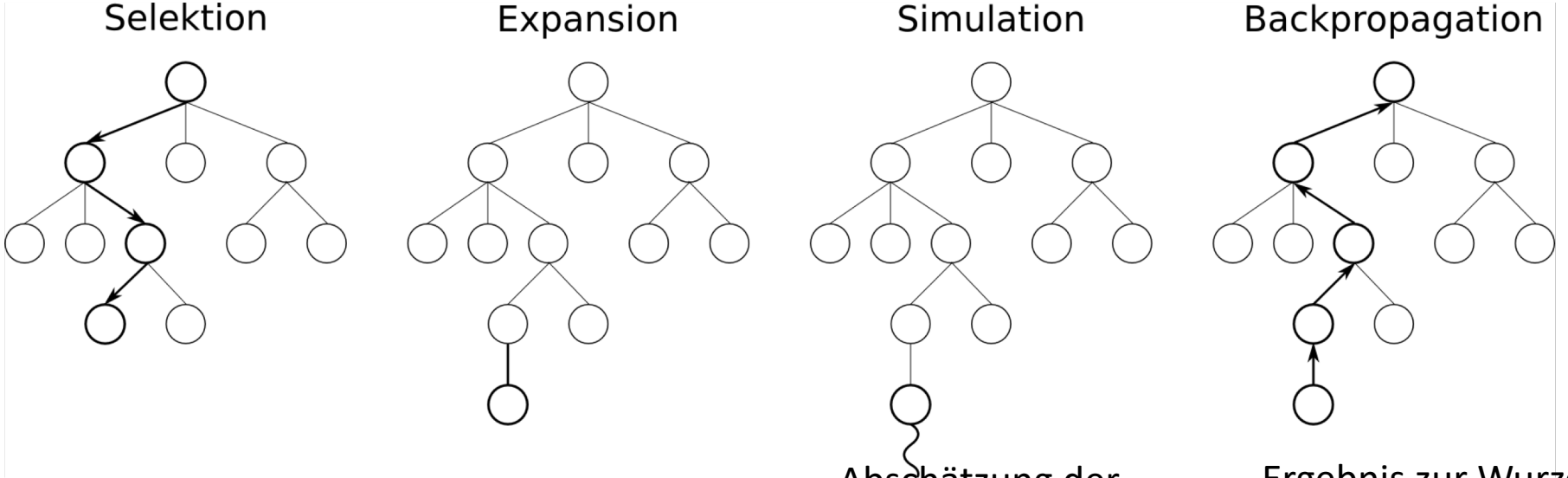
Monte-Carlo Tree Search

- Idee: Erweitere Entscheidungsbaum durch stichprobenartige Simulationen mit zufälligen Zügen



Monte-Carlo Tree Search

- Problem: Welche Knoten/Pfade sollen weiter iteriert werden?
  - Exploitation: Wahl der Knoten mit hoher Gewinnrate
  - Exploration: Wahl der Knoten mit wenigen Simulationen



Wahl eines Pfades, der abgeschätzt werden soll

Wahl eines weiteren Zuges

Abschätzung der Punkte mittels zufälliger Züge

Ergebnis zur Wurzel propagieren

Exploration vs Exploitation

- Problem: Welche Knoten/Pfade sollen weiter iteriert werden?
  - Exploitation: Wahl der Knoten mit hoher Gewinnrate
  - Exploration: Wahl der Knoten mit wenigen Simulationen

- Upper Confidence Bound
  An Knoten n wähle Nachfolger n_i, für den $\frac{w_i}{\mathrm{n}_i+\varepsilon} + c\sqrt{\frac{\ln \mathrm{N_i}}{\mathrm{n_i}+\varepsilon}}$ maximal ist
  - w_i: Zu erwartender Punktestand ab Knoten n_i
  - $\mathrm{n_i}$: Anzahl bisher durchgeführter Simulationen ab Knoten n_i
  - c: Erkundungs-Parameter (z.B. $\sqrt{2}$)
  - N_i: Gesamtanzahl der bisher durchgeführter Simulationen nach dem i-ten Zug

- Erste Komponente: Exploitation (hoch, wenn Gewinnwahrscheinlichkeit hoch ist)
- Zweite Komponente: Exploration (hoch, wenn wenig Simulationen dort gemacht wurden)

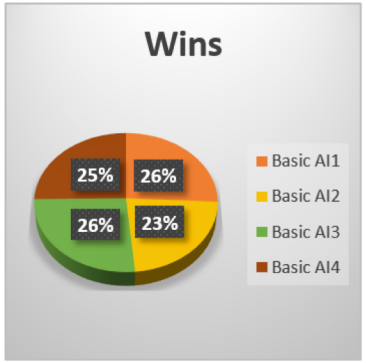
Monte-Carlo Tree Search

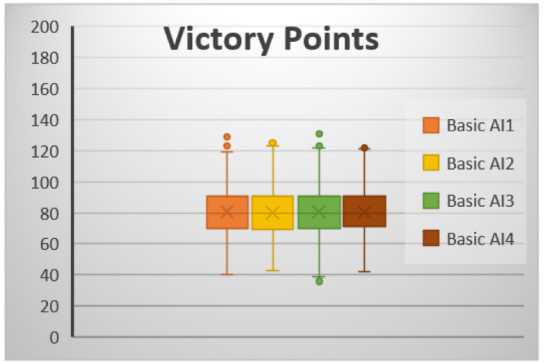
- Vorteil:
  - Monte Carlo Tree Search konvergiert (sehr langsam) gegen MiniMax
  - Es muss nicht der beste Zug in einer Position bestimmt werden, sondern nur erlaubte Züge generiert werden
  - Implementierungen in verschiedene Sprachen vorhanden (Python, Java, C#)

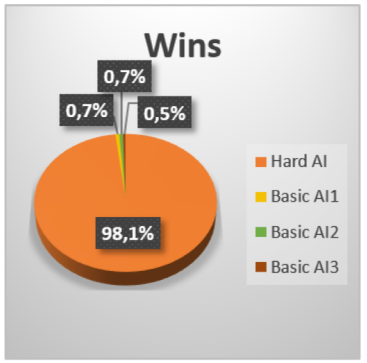
- Nachteil:
  - Einzelne Züge, die in dieser Stellung zur Niederlage führen, werden evtl. nicht gefunden

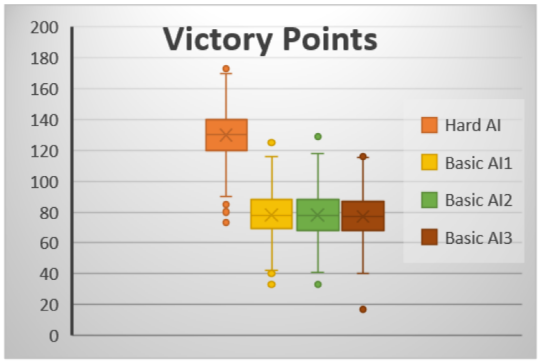
Workflow

- Batch-Modus:
  - Neue KIs wiederholt (zB 1000 Mal) gegen alte KIs antreten lassen
  - Messen der Siegpunkte
- Workflow: Spiel gegen KI
  - Neue Ideen (KIs) implementieren
  - Test im Batchmodus
  - Änderungen ggf. übernehmen









https://connect.unity.com/p/articles-boardgame-ai

Mögliche Strategien

- Destination Hungry Agent (DHA)
  - Ziehe maximale Anzahl überlappender Zielkarten bis Anzahl Train-Tokens erreicht wurde
  - Berechne Steiner Tree
  - Baue Strecken

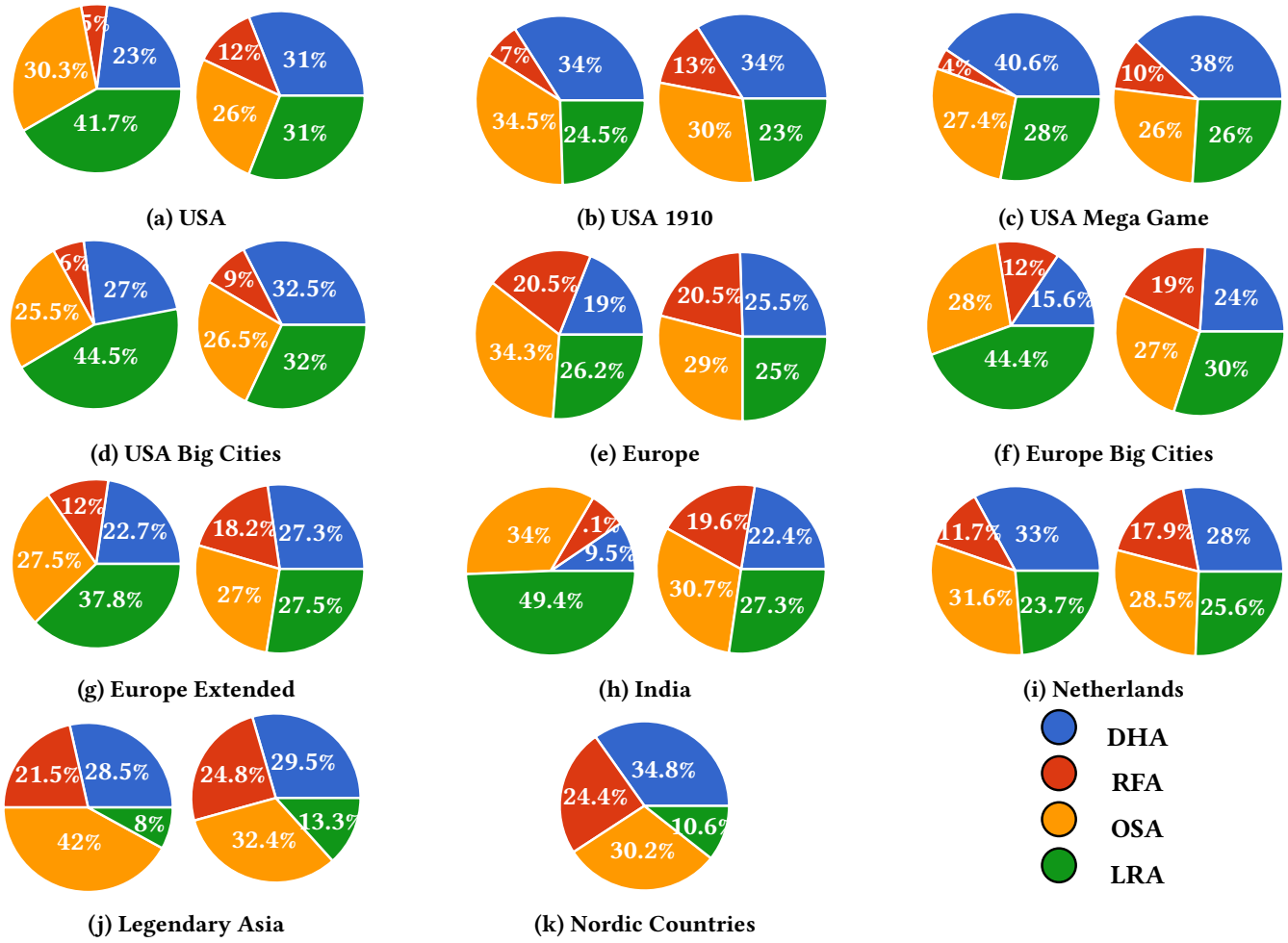
- Route Focused Agent (RFA)
  - Behalte alle 3 initialen Zielkarten
  - Erfülle alle Zielkarten
  - Baue anschließend lange Strecken

Mögliche Strategien

- One Step Agent (OSA)
  - Ziehe möglichst überlappende Zielkarten
  - Sobald Zielkarten erfüllt sind, ziehe neue Zielkarten

- Long Route Agent (LRA)
  - Behalte alle initialen Zielkarten
  - Aber: Baue nur Strecken der Länge >= 3
  - Sobald alle Zielkarten erfüllt wurden: baue nur noch lange Strecken

Welche Strategien funktionieren gut?



Destination Hungry Agent (DHA)
First keep drawing tickets,
then build Steiner Tree

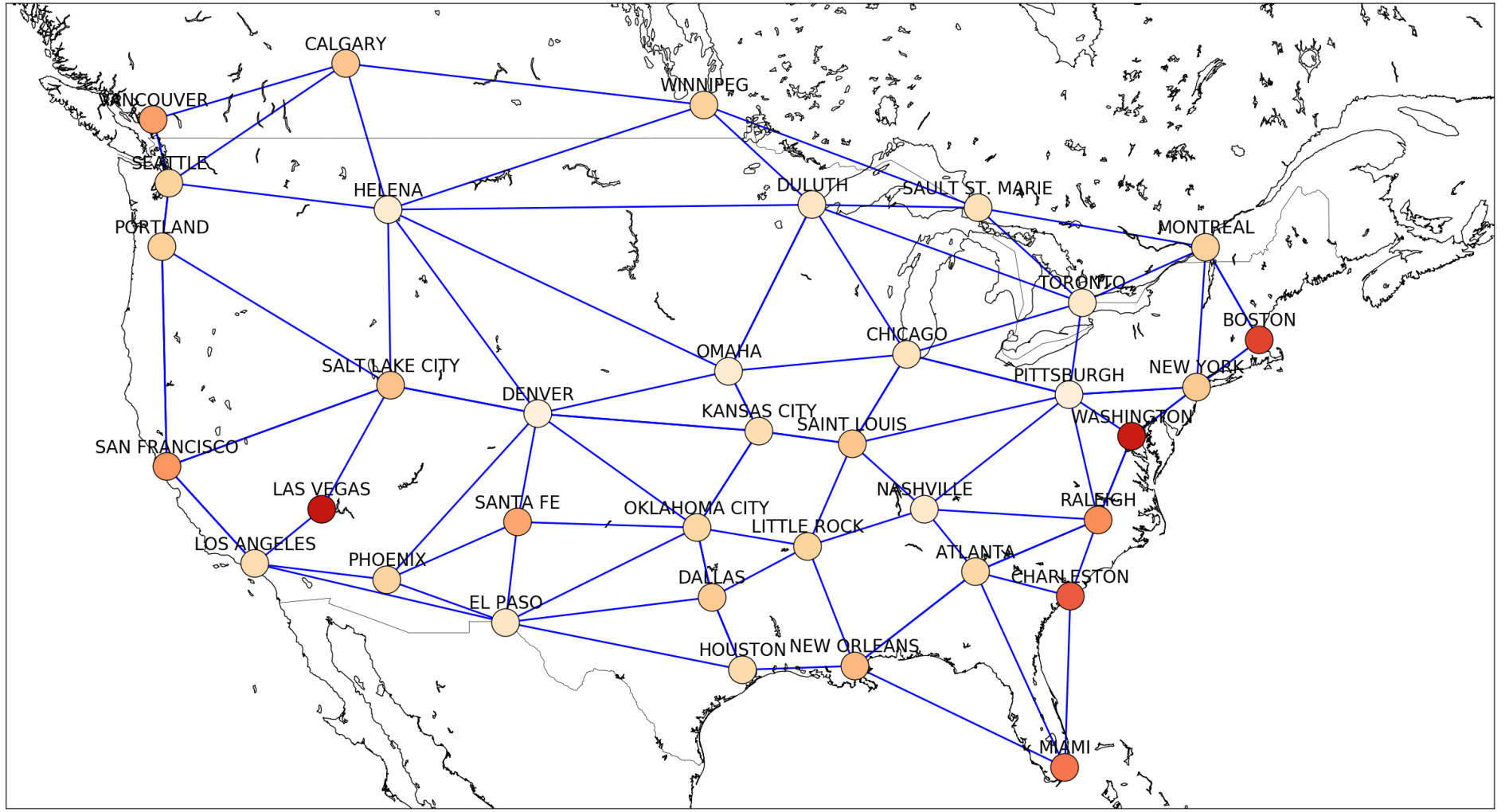
Route Focused Agent (RFA)
Fulfill all initial tickets,
then build long routes

One Step Agent (OSA)
First fulfill held tickets,
then draw new tickets

Long Route Agent (LRA)
First fulfill held tickets,
but Focus on building long
(>=3) routes

http://julian.togelius.com/Silva2017AI.pdf

Welche Städte wurden bei Siegen am öftesten bebaut



Red = undesirable

http://julian.togelius.com/Silva2017AI.pdf

Weiterführende Links

- Vorlesung „General Game Playing“ von Sebastian Wandelt im WS 13/14: https://hu.berlin/vl_game
- Ticket To Ride Strategies: http://julian.togelius.com/Silva2017AI.pdf
- Ticket to Ride in Java mit implementierter (aber unkommentierter) Minimax- und Monte-Carlo-Baumsuche https://github.com/cooijmanstim/hobo
- Reddit-Thread mit allerlei nützlichen Informationen und Erfahrungswerten: https://www.reddit.com/r/gameai/comments/44l6w7/building_an_ai_for_a_multiplayer_turnbased_board/

KI-Challenge: Ablauf

- Einwöchige Turniere: Erste lauffähige KI zum 14.1.; wöchentliche Verbesserungen
- Wöchentliches 1on1-Turnier (4 Spielpaarungen)
  - Zunehmende Punkte für erfolgreiche Teilnahme und Platzierung
    - 14.1.: 1-1-2, 21.1.: 1-2-3, 28.1.: 2-3-5, 4.2.: 3-5-8, 11.2.: 5-8-13
  - 0 Punkte, wenn KI wiederholt abstürzt oder Spiel nicht terminiert
  - Wir stellen Rechner für Spielserver und visualisieren exemplarisches Spiel
  - Turnierteilnehmer senden vorab ihre KI an uns
- Zusätzliches Turnier für Einzelabgaben
  - K.O.-System; Beginn im Halbfinale
  - vorzugsweise (aber nicht zwingend) 1on1
  - Live jeden Montag (Laptops verbinden sich mit Server)